Bitcoin: Un Sistema de Efectivo Electrónico Usuario-a-Usuario

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Traducido al Español de bitcoin.org/bitcoin.pdf
por Angel León - www.diariobitcoin.com

Abstracto. Una versión puramente electrónica de efectivo permitiría que los pagos en línea fuesen enviados directamente de un ente a otro sin tener que pasar por medio de una institución financiera. Firmas digitales proveen parte de la solución, pero los beneficios principales se pierden si existe un tercero confiable para prevenir el doble-gasto. Proponemos una solución al problema del doble gasto utilizando una red usuario-a-usuario. La red coloca estampas de tiempo a las transacciones al crear un hash de las mismas en una cadena continua de pruebas de trabajo basadas en hashes, formando un registro que no puede ser cambiado sin volver a recrear la prueba de trabajo. La cadena más larga no solo sirve como la prueba de la secuencia de los eventos testificados, sino como prueba de que vino del gremio de poder de procesamiento de CPU más grande. Siempre que la mayoría del poder de procesamiento de CPU esté bajo el control de los nodos que no cooperan para atacar la red, estos generarán la cadena más larga y le llevarán la ventaja a los atacantes. La red en sí misma requiere una estructura mínima. Los mensajes son enviados bajo la base de mejor esfuerzo, y los nodos pueden irse y volver a unirse a la red como les parezca, aceptando la cadena de prueba de trabajo de lo que sucedió durante su ausencia.

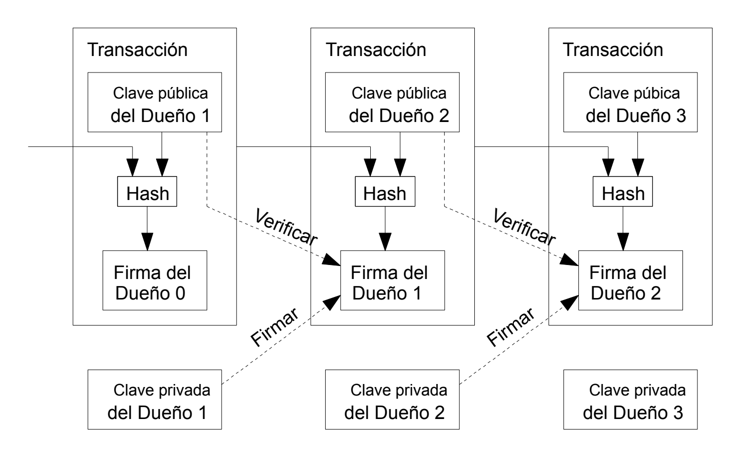
1. Introducción

El comercio en el Internet ha venido a depender exclusivamente de instituciones financieras las cuales sirven como terceros confiables para el procesamiento de pagos electrónicos. Mientras que el sistema funciona lo suficientemente bien para la mayoría de las transacciones, aún sufre de las debilidades inherentes del modelo basado en confianza. Transacciones completamente no-revertibles no son realmente posibles, dado que las instituciones financieras no pueden evitar mediar disputas. El costo de la mediación incrementa costos de transacción, limitando el tamaño mínimo práctico por transacción y eliminando la posibilidad de pequeñas transacciones casuales, y hay un costo más amplio en la pérdida de la habilidad de hacer pagos no-reversibles por servicios no-reversibles. Con la posibilidad de revertir, la necesidad de confianza se expande. Los comerciantes deben tener cuidado de sus clientes, molestándolos pidiendo más información de la que se necesitaría de otro modo. Un cierto porcentaje de fraude es aceptable como inevitable. Estos costos e incertidumbres de pagos pueden ser evitadas en persona utilizando dinero físico, pero no existe un mecanismo para hacer pagos por un canal de comunicación sin un tercero confiable.

Lo que se necesita es un sistema de pagos electrónicos basado en pruebas criptográficas en vez de confianza, permitiéndole a dos partes interesadas en realizar transacciones directamente sin la necesidad de un tercero confiable. Las transacciones que son computacionalmente poco factibles de revertir protegerían a los vendedores de fraude, y mecanismos de depósitos de fideicomisos de rutina podrían ser fácilmente implementados para proteger a los compradores. En este trabajo, proponemos una solución al problema del doble-gasto utilizando un servidor de marcas de tiempo usuario-a-usuario distribuido para generar una prueba computacional del orden cronológico de las transacciones. El sistema es seguro mientras que nodos honestos controlen colectivamente más poder de procesamiento (CPU) que cualquier grupo de nodos atacantes en cooperación.

2. Transacciones

Definimos una moneda electrónica como una cadena de firmas digitales. Cada dueño transfiere la moneda al próximo al firmar digitalmente un hash de la transacción previa y la clave publica del próximo dueño y agregando estos al final de la moneda. Un beneficiario puede verificar las firmas para verificar la cadena de propiedad.

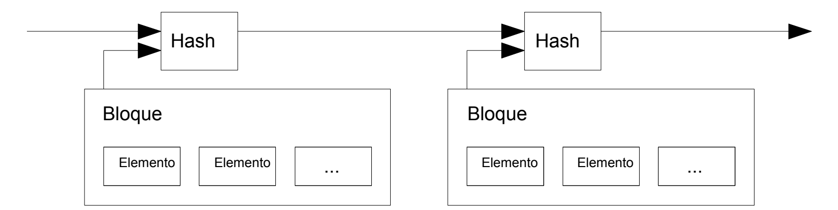


El problema claro es que el beneficiario no puede verificar si uno de los dueños no se hizo un doble-gasto de la moneda. Una solución común es introducir una autoridad central confiable, o casa de moneda, que revisa cada si cada transacción tiene doble-gasto. Después de cada transacción, la moneda debe ser regresada a la casa de moneda para generar una nueva moneda, y solo las monedas generadas directamente de la casa de moneda son las que se confían de no tener doble-gasto. El problema con esta solución es que el destino del sistema monetario entero depende de la compañía que gestiona la casa de moneda, con cada transacción teniendo que pasar por ellos, tal como un banco.

Necesitamos una forma para que el beneficiario pueda saber que los dueños previos no firmaron ningunas transacciones más tempranas. Para nuestros propósitos, la transacción más temprana es la que cuenta, así que no nos importan otros intentos de doble-gasto más tarde. La única forma de confirmar la ausencia de una transacción es estando al tanto de todas las transacciones. En el modelo de la casa de moneda, la casa de moneda estaba al tanto de todas las transacciones y esta decidiría cuales llegaban primero. Para lograr esto sin un tercero confiable, las transacciones deben ser anunciadas públicamente [1], y necesitamos un sistema de participantes que estén de acuerdo con una historia única del orden en que estas fueron recibidas. El beneficiario necesita prueba de que a la hora de cada transacción, la mayoría de los nodos estuvieron de acuerdo que esta fue la primera que se recibió.

3. Servidor de marcas de tiempo.

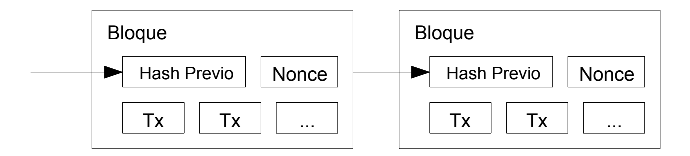
La solución que proponemos comienza con un servidor de marcas de tiempo. Un servidor de marcas de tiempo funciona al tomar un hash de un bloque de elementos a ser fechados y publicando ampliamente el hash, tal como en un periódico, o una publicación Usenet [2-5]. La marca de tiempo prueba que la data debe haber existido en el tiempo, obviamente, para meterse dentro del hash. Cada marca de tiempo incluye la marca de tiempo previa en su hash, formando una cadena, con cada marca de tiempo adicional reforzando las anteriores a esa.



4. Prueba-de-trabajo

Para implementar un servidor de marcas de tiempo en una base usuario-a-usuario, necesitaremos utilizar un sistema de prueba-de-trabajo similar al Hashcash de Adam Back [6], en vez de un periódico o una publicación en Usenet. La prueba-de-trabajo envuelve la exploración de un valor que al calcular un hash, tal como SHA-256, el hash empiece con un número de bits en cero. El trabajo promedio requerido es exponencial en el número de bits puestos en cero requeridos y puede ser verificado ejecutando un solo hash.

Para nuestra red de marcas de tiempo, implementamos la prueba-de-trabajo incrementando un nonce en el bloque hasta que un valor es encontrado que de el número requerido de bits en cero para el hash del bloque. Una vez que el esfuerzo de CPU se ha gastado para satisfacer la prueba-de-trabajo, el bloque no puede ser cambiado sin rehacer todo el trabajo. A medida que más bloques son encadenados después de este, el trabajo para cambiar el bloque incluiría rehacer todos los bloques después de este.



La prueba-de-trabajo también resuelve el problema de determinar la representación en cuanto a decisión por mayoría. Si la mayoría fuese basada en un voto por dirección IP, podría ser subvertida por alguien capaz de asignar muchos IPs. Prueba-de-trabajo es esencialmente un-CPU-un-voto. La decisión de la mayoría es representada por la cadena más larga, la cual tiene la prueba-de-trabajo de mayor esfuerzo invertido en ella. Si la mayoría del poder de CPU es controlada por nodos honestos, la cadena honesta crecerá más rápido y pasará cualquier cadena que esté compitiendo. Para modificar un bloque en el pasado, un atacante tendría que rehacer la prueba-de-trabajo del bloque y de todos los bloques después y luego alcanzar y pasar el trabajo de los nodos honestos. Luego demostraremos que la probabilidad de un atacante más lento de alcanzar disminuye exponencialmente a medida que bloques subsecuentes son añadidos.

Para compensar por el incremento de velocidad de hardware y en el interés variante de corre nodos en el tiempo, la dificultad de la prueba-de-trabajo es determinada por una media móvil dirigida a un número promedio de bloques por hora. Si estos se generan muy rápido, la dificultad incrementa.

5. La Red

Los pasos para gestionar la red son como sigue:

1) Transacciones nuevas son emitidas a todos los nodos.
2) Cada nodo recolecta nuevas transacciones en un bloque.
3) Cada nodo trabaja en encontrar una prueba-de-trabajo difícil para su bloque.
4) Cuando un nodo encuentra una prueba-de-trabajo, emite el bloque a todos los nodos.
5) Los nodos aceptan el bloque si todas las transacciones en el bloque son válidas y no se han gastado ya.
6) Los nodos expresan su aceptación del bloque al trabajar en crear el próximo bloque en la cadena, utilizando el hash del bloque aceptado como el hash previo.

Los nodos siempre consideran la cadena más larga como la correcta y empiezan a trabajar en extenderla. Si dos nodos emiten versiones diferentes del próximo bloque simultáneamente, algunos nodos puede que reciban uno o el otro primero. En ese caso, trabajan en el primero que reciban pero guardan la otra rama en caso de que esta se vuelva más larga. El empate se rompe cuando la próxima prueba-de-trabajo es encontrada y una rama se vuelve más larga; los nodos que estaban trabajando en la otra rama luego se cambian a la más larga.

Las emisiones de nuevas transacciones no necesariamente necesitan llegar a todos los nodos. Tanto estas lleguen a muchos nodos, entrarán a un bloque antes de que pase mucho tiempo. Las emisiones de bloques también son tolerantes a mensajes perdidos. Si un nodo no recibe un bloque, lo va a pedir cuando reciba el próximo bloque y se de cuenta que se perdió uno.

6. Incentivo

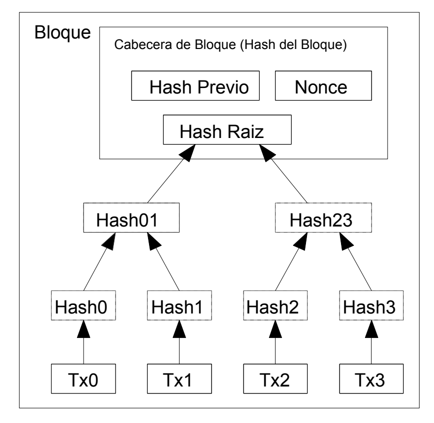
Por convención, la primera transacción en el bloque es una transacción especial que comienza una moneda nueva cuyo dueño es el creador del bloque. Esto agrega un incentivo para que los nodos apoyen a la red, y provee una forma inicial de distribuir monedas en circulación, dado que no hay una autoridad para crearlas. Esta adición estable de una cantidad constante de monedas nuevas es análoga a mineros de oro gastando recursos para agregar oro a la circulación. En nuestro caso, es el tiempo del CPU y la electricidad que se gasta.

El incentivo también puede ser fundado con costos de transacción. Si el valor de salida de una transacción es menor que la entrada, la diferencia es una tarifa de transacción que se le añade al valor de incentivo del bloque que contiene la transacción. Una vez que un número predeterminado de monedas han entrado en circulación, el incentivo puede transicionar enteramente a tarifas de transacción y ser completamente libre de inflación.

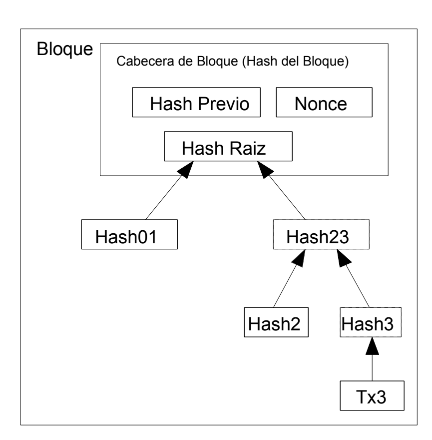
El incentivo puede ayudar a animar a los nodos a mantenerse honestos. Si un atacante egoísta es capaz de reunir más potencia de CPU que todos los nodos honestos, este tendría que elegir entre utilizarla para defraudar a la gente robando sus pagos de vuelta, o en utilizarla para generar monedas nuevas. Debería encontrar más rentable jugar por las reglas, tales regla lo favorecen a el con más monedas que a todos los demás combinados, que socavar el sistema y la validez de su propia riqueza.

7. Reclamando Espacio en Disco

Una vez que la última transacción en una moneda es enterrada bajo suficientes bloques, las transacciones gastadas antes de estas pueden ser descartadas para ahorrar espacio en disco. Para facilitar esto sin romper el hash del bloque, las transacciones se les comprueba en un árbol Merkle [7] [2] [5], con la única raíz incluida en el hash el bloque. Los bloques viejos pueden ser compactados al sacar ramas del árbol. Los hashes interiores no necesitan ser guardados.



Transacciones *Hasheadas* en un árbol Merkle

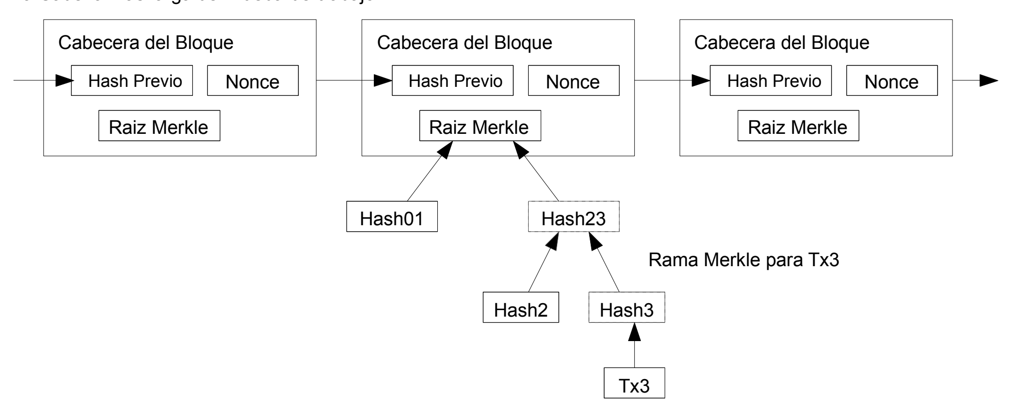


Después de podar Tx0-2 del Bloque

La cabecera de un bloque sin transacciones sería de unos 80 bytes. Si suponemos que cada bloque es generado cada 10 minutos, 80 bytes * 6 * 24 * 365 = 4.2MB por año. Con computadoras generalmente vendiéndose con 2GB de RAM para el 2008, y la ley de Moore prediciendo el crecimiento actual de 1.2GB por año, el almacenamiento no debe ser un problema aun si las cabeceras de los bloques deben permanecer en memoria.

8. Verificación de Pagos Simplificada

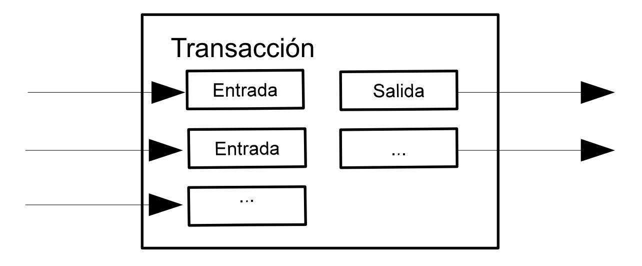
Es posible verificar pagos sin correr un nodo de red completo. Un usuario solo necesita mantener una copia de las cabeceras de los bloques de la cadena más larga de prueba-de-trabajo, la cual puede obtener haciendo una búsqueda en los nodos de red hasta que esté convencido que tenga la cadena más larga, y obtenga la rama Merkle que enlaza la transacción al bloque en que ha sido fechado. No puede verificar la transacción por sí mismo, pero al enlazarla a un lugar en la cadena, ahora puede ver que un nodo de la red la ha aceptado y los bloques añadidos después confirman aún más que la red lo ha aceptado.



Como tal, la verificación es confiable a medida que nodos honestos controlen la red, pero es más vulnerable si la red es dominada por un atacante. Mientras que los nodos de la red puedan verificar transacciones por si mismos, el método simplificado puede ser engañado por las transacciones fabricadas de un atacante hasta que el atacante pueda continuar dominando la red. Una estrategia para protegerse de esto es aceptar alertas de los nodos de la red cuando detecten un bloque inválido, pidiéndole al usuario que se baje el bloque completo y las transacciones alertadas para confirmar la inconsistencia. Los negocios que reciban pagos frecuentes van a querer correr sus propios nodos para seguridad más independiente y verificación más rápida.

9. Combinando y Dividiendo Valor

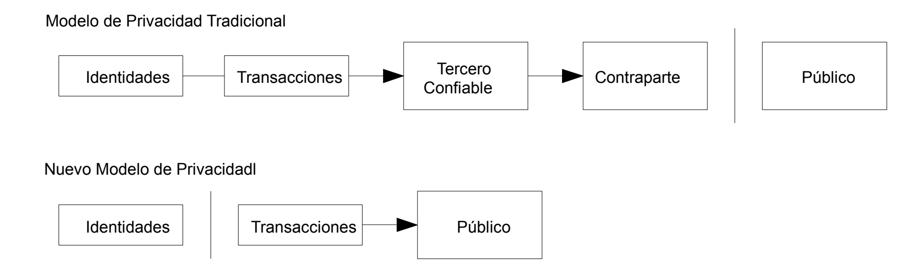
Aunque sería posible manipular monedas individualmente, seria difícil de manejar el hacer una transacción por cada centavo en una transferencia. Para permitir que el valor se divida y se combine, las transacciones contienen múltiples entradas y salidas. Normalmente habrán o una sola entrada de una transacción previa más grande o múltiples entradas combinando cantidades más pequeñas, y al menos dos salidas: una para el pago, y una para devolver el cambio, si es que hay algún cambio, de vuelta al emisor.



Debe ser notado que donde una transacción depende de varias transacciones, y esas transacciones dependen en muchas más, no hay ningún problema. Nunca existe la necesidad de extraer una copia completa de la transacción por si sola de la historia de transacciones.

10. Privacidad

El modelo bancario tradicional logra un nivel de privacidad al limitar el acceso a la información de las partes envueltas y del tercero confiado. La necesidad de anunciar todas las transacciones públicamente se opone a este método, pero la privacidad aún puede ser mantenida al romper el flujo de la información en otro lugar: al mantener las claves públicas anónimas. El público puede ver que alguien está enviando una cantidad a otra persona, pero sin información que relacione la transacción a ninguna persona. Esto es similar al nivel de información mostrado por las bolsas de valores, donde el tiempo y el tamaño de las transacciones individuales, la "cinta", es público, pero sin decir quienes son las partes.



Como un cortafuegos adicional, un par nuevo de claves debe ser utilizado para cada transacción de modo que puedan ser asociadas a un dueño en común. Algún tipo de asociación es inevitable con transacciones de múltiples entradas, las cuales pueden revelar que sus entradas fueron apropiadas por el mismo dueño. El riesgo está en que si el dueño de una clave es revelado, el enlazado podría revelar otras transacciones que pertenecieron al mismo dueño.

11. Cálculos

Consideramos el escenario en el que un atacante intenta generar una cadena alterna más rápido que la cadena honesta. Aún si esto es logrado, esto no abre el sistema a cambios arbitrarios, tal como crear valor del aire o tomar dinero que nunca le perteneció al atacante. Los nodos no aceptarían una transacción inválida como pago, y los nodos honestos nunca aceptará un bloque que las contenga. Un atacante puede únicamente intentar cambiar solo una de sus propias transacciones para retomar dinero que ha gastado recientemente.

La carrera entre una cadena honesta y la cadena de un atacante puede ser caracterizada como una Caminata Aleatoria Binomial. El evento de éxito es la cadena honesta siendo extendida por un bloque, incrementar esta ventaja por +1, y el evento de fracaso es la cadena del atacante siendo extendida por un bloque reduciendo la distancia por -1.

La probabilidad de que un atacante pueda alcanzar desde un déficit dado es análogo al problema de la Ruina del Apostador. Supóngase que un apostador con crédito ilimitado empieza en un déficit y juega potencialmente un número infinito de intentos para intentar llegar a un punto de equilibrio. Podemos calcular la probabilidad de que llegase al punto de equilibrio, o que un atacante llegue a alcanzar a la cadena honesta, como sigue [8]:

p = probabilidad de que un nodo honesto encuentre el próximo bloque
q = probabilidad de que el atacante encuentre el próximo bloque
q_z = probabilidad de que el atacante llegue a alcanzar desde z bloques atrás.

$$q_z = \begin{Bmatrix} 1 & \text{if } p \leq q \\ (q/p)^z & \text{if } p > q \end{Bmatrix}$$

Dada nuestra hipótesis de que p > q, la probabilidad cae exponencialmente mientras que el número de bloques el cual el atacante debe alcanzar incrementa. Con las probabilidades en contra, si no hace una estocada afortunada desde el principio, sus chances se vuelven extremadamente pequeños a medida que se queda más atrás.

Ahora consideramos cuánto necesita esperar el recipiente de una nueva transacción antes de tener la certeza suficiente de que el emisor no puede cambiar la transacción. Asumimos que el emisor es un atacante el cual quiere hacerle creer al recipiente que le pagó durante un rato, luego cambiar la transacción para pagarse de vuelta a sí mismo una vez que ha pasado un tiempo. El receptor será alertado cuando esto suceda, pero el emisor espera que sea demasiado tarde.

El receptor genera una nuevo par de claves y entrega la clave pública al emisor poco después de hacer la firma. Esto previene que el emisor prepare una cadena de bloques antes de tiempo al trabajar continuamente hasta que tenga la suerte de adelantarse lo suficiente, y luego ejecutar la transacción en ese momento. Una vez que la transacción es enviada, el emisor deshonesto empieza a trabajar en secreto en una cadena paralela que contiene una versión alterna de su transacción.

El recipiente espera a que la transacción sea añadida al bloque y z bloques han sido enlazados después de la transacción. El no necesita saber la cantidad exacta de progreso que al atacante ha logrado, pero asumiendo que los bloques honestos se tardaron el promedio esperado por bloque, el progreso potencial del atacante será una distribución de Poisson con un valor esperado:

$$\lambda = z\frac{q}{p}$$

Para obtener la probabilidad de que el atacante aún pueda alcanzar ahora, multiplicamos la densidad de Poisson por cada cantidad de progreso que pudo haber hecho por la probabilidad de que pudo alcanzar desde ese punto:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{Bmatrix} (q/p)^{(z-k)} & if\, k \le z \\ 1 & if\, k > z \end{Bmatrix}$$

Re-organizamos para evitar la suma de la cola infinita de la distribución...

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

Convertimos a código en C...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```

Ejecutamos algunos resultados, podemos ver que la probabilidad cae exponencialmente con z.

```
q=0.1
z=0    P=1.0000000
z=1    P=0.2045873
z=2    P=0.0509779
z=3    P=0.0131722
z=4    P=0.0034552
z=5    P=0.0009137
z=6    P=0.0002428
z=7    P=0.0000647
z=8    P=0.0000173
z=9    P=0.0000046
z=10   P=0.0000012

q=0.3
z=0    P=1.0000000
z=5    P=0.1773523
z=10   P=0.0416605
z=15   P=0.0101008
z=20   P=0.0024804
z=25   P=0.0006132
z=30   P=0.0001522
z=35   P=0.0000379
z=40   P=0.0000095
z=45   P=0.0000024
z=50   P=0.0000006
```

Resolvemos para P menor que 0.1%...

```
P < 0.001
q=0.10   z=5
q=0.15   z=8
q=0.20   z=11
q=0.25   z=15
q=0.30   z=24
q=0.35   z=41
q=0.40   z=89
q=0.45   z=340
```

12. Conclusión

Hemos propuesto un sistema para transacciones electrónicas sin depender en confianza. Comenzamos con el marco habitual de monedas hechas de firmas digitales, el cual provee un control fuerte de propiedad, pero es incompleto sino existe una forma de prevenir doble-gasto. Para solucionar esto, hemos propuesto una red usuario-a-usuario que utiliza prueba-de-trabajo para registrar una historia pública de transacciones la cual rápidamente se convierte impráctica computacionalmente para que un atacante pueda cambiar si nodos honestos controlan la mayoría del poder de CPU. La red es robusta en su simplicidad no estructurada. Los nodos pueden trabajar todos al mismo tiempo con poca coordinación. No necesitan ser identificados, dado que los mensajes no son enrutados a ningún lugar en particular y solo necesitan ser entregados bajo la base de un mejor esfuerzo. Los nodos pueden irse y volver a la red a voluntad, aceptando la cadena de prueba-de-trabajo como prueba de lo que sucedió mientras estuvieron ausentes. Votan con su poder de CPU, expresando su aceptación de los bloques válidos al trabajar extendiéndose y rechazando bloques inválidos al refutar trabajar en ellos. Cualquier reglas necesarias e incentivos se pueden hacer cumplir con este mecanismo de consenso.

Referencias

[1] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

[2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.

[3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3, no 2, pages 99-111, 1991.

[4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," *In Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.

[5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 28-35, April 1997.

[6] A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.

[7] R.C. Merkle, "Protocols for public key cryptosystems," In Proc. *1980 Symposium on Security and Privacy*, IEEE Computer Society, pages 122-133, April 1980.

[8] W. Feller, "An introduction to probability theory and its applications," 1957.